REMARKS/ARGUMENTS

Claims 1-20 are pending in the present application. Claims 7 and 17 were canceled; claims 1-6, 8, 10-16, 19 and 20 were amended. Support for the claim amendments can be found in claims 7 and 17 as originally filed, and on page 26 of the specification. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 101: Claims 11-20

The examiner has rejected claims 11-20 under 35 U.S.C. § 101 as being directed towards non-statutory subject matter. Claims 11 and 20 have been amended accordingly, thus overcoming the Examiner's rejection.

II. 35 U.S.C. § 102, Anticipation: Claims 1, 4-8, 10-11, and 14-20

The examiner has rejected claims 1, 4-8, 10-11, and 14-20 under 35 U.S.C. § 102 as being anticipated by *Carlson*, System and Method for Caching Javaserver Pages™ Responses, U.S. Patent No. 6,697,849 (February 24, 2004) (hereinafter "*Carlson*"). This rejection is respectfully traversed.

With regard to claim 1, the Examiner states:

> 6. Referring to claim 1, Carlson discloses a method of distributing traffic to application instances (i.e. applications 202-208 running on application server 200) on one or more computing devices (i.e. servers 308A-C), comprising:
>
> obtaining application instance specific operational information (i.e. server load criteria and application component performance criteria) identifying operational characteristics (i.e. elements shown in Figures 8 and 9) of an application instance on a computing device on the one or more computing devices (e.g. abstract; col. 12, lines 40-67);
>
> generating a load balancing weight to be associated with an application instance based on the application instance specific operational information (i.e. random number is generated in a weighted manner according to the "best" server at that particular time) (col. 16, lines 13-47); and
>
> distributing traffic based on the generated load balancing weight (i.e. "gracefully" distribute requests among the application servers) (col. 16, lines 35-47).

Office Action dated March 30, 2007, p. 3

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-*

Clark Corp., 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Claim 1, as amended, is as follows:

> 1. A method, in a data processing system, of distributing traffic to application instances on one or more computing devices, comprising:
>
> obtaining application instance specific operational information identifying operational characteristics of an application instance on a computing device of the one or more computing devices, wherein the application instance specific operational information includes at least one of a number of successful transactions processed by the application instance within a period of time, an application instance topology, an importance of transactions currently being processed by the application instance, an amount of time the application instance has been blocked waiting for resources, and an amount of resources consumed by the application instance;
>
> generating a load balancing weight to be associated with the application instance based on the application instance specific ~~operation~~ operational information obtained; and
>
> distributing the traffic to the application instance based on the ~~generated~~ load balancing weight.

Carlson does not anticipate claim 1 as amended, because *Carlson* does not teach "obtaining application instance specific operational information identifying operational characteristics of an application instance on a computing device of the one or more computing devices, *wherein the application instance specific operational information includes at least one of a number of successful transactions processed by the application instance within a period of time, an application instance topology, an importance of transactions currently being processed by the application instance, an amount of time the application instance has been blocked waiting for resources, and an amount of resources consumed by the application instance....*" *Carlson* teaches load balancing across servers according to the following:

> One general approach which may be used in selecting an application server to send a request to is to leave the decision to the client. The client may keep track of the response times seen over time from various application servers and may choose to send requests to the application server with the historically fastest response times. In many cases, the "client" of an application server is a web server. As shown in FIG. 4, a web server may have a web server plug-in which includes a load balancer component or module. This load balancer component may be responsible for monitoring which application servers are available in a cluster to service requests, may record the response times seen for requests serviced by each application server, and may use this information to determine the most appropriate application server to send a given request to.

Carlson, col. 11, ll. 13-27

> The client, e.g., the load balancing component of the web server plug-in, may also make load balancing decisions based on factors other than response times. For example, in one embodiment, administrators may assign a "weight" to each application server in a cluster, using an administrative tool. A weight may be assigned to each

application server based on the server's resources, such as the number of CPUs, the memory capacity, etc. The application server weights may then be used in various request distribution algorithms, such that requests are distributed among the application servers in proportion to their weights. For example, weights may be used in a weighted round-robin algorithm or may be applied to enforce even distribution for certain types of requests, as described below.

Carlson, col. 11, ll. 13-27

With the exception of server response time, each of the measured metrics that *Carlson* discloses are the type of general system based information that the Applicant defines in the specification. This general system based information is not the application instance specific operational information claimed. General system statistics information include system central processing unit (CPU) utilization, available system memory, available disk space, number of connections, and the like. These general system statistics are obtained by the agent application from the operating system layer and are sent to the load balancing device for use in determining how to perform load balancing of incoming requests directed to an application, of which application instance is one instance.

The problem with using general system statistics, as previously noted above, is that they may not accurately reflect the load on a particular application running on the system for which the general system statistics are obtained. For example, while the CPU utilization may be low, and thus it would seem that this server computing device should be preferred over others when performing load balancing, the bottleneck in processing transactions may be in the input/output (I/O) system or a network interface card. Similarly, just because the CPU utilization may be high does not necessarily mean that the system should be unavailable for more incoming requests, i.e. traffic. If all of the transactions being processed on a server computing system with high CPU utilization are of a low priority, the load balancing device may want to send a high priority job or request to this server computing system since the high priority job will preempt the low priority transactions and will be processed first.

The present application solves the problems associated with using general system statistics in the manner used in the known methodologies by instrumenting application instances to communicate with agent applications and provide application instance specific operational information which may then be used to calculate or adjust weights applied to these server/application instance pairs during load balancing.

Claim 1 has been amended to further clarify the distinctions between the general system statistics disclosed in *Carlson* and the application instance specific operational information claimed in claim 1. Support for the claim amendments can be found in claim 7 of the application. Claim 1, as amended, excludes "application instance response time," originally included in claim 7. Thus, again, *Carlson* does not teach each feature of amended claim 1.

Independent claims 11 and 20 have been amended consistent with the amendments to claim 1. Claims 4-8 and 10 depend from claim 1. Claims 14-19 depend from claim 11. Therefore, the same distinctions between *Carlson* and claim 1 apply to the respective dependent claims. Therefore, the rejection of claims 1, 4-8, 10-11, and 14-20 under 35 U.S.C. § 102 has been overcome.

III. <u>35 U.S.C. § 103, Obviousness: Claim 9</u>

The examiner has rejected claim 9 under 35 U.S.C. § 103 as being unpatentable over *Carlson*. This rejection is respectfully traversed.

With regard to claim 9, the Examiner states:

> 14. Carlson discloses the invention substantively as described in claim 1. Carlson does not explicitly state that the weighting is done separately from the computing devices or a load balancing device, however it has been held obvious to make parts separable. See Nerwin v. Erlichman 168 USPQ 177 (1969). By this rationale, one of ordinary skill in the art would have found it obvious to separate the weight management system from a load balancing device or the computing devices in order to reduce processing overhead, thereby resulting in a more efficient system.

Office Action dated March 30, 2007, p. 5

The Examiner bears the burden of establishing a *prima facie* case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). The scope and content of the prior art are... determined; differences between the prior art and the claims at issue are... ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background the obviousness or non-obviousness of the subject matter is determined. *Graham v. John Deere Co.*, 383 U.S. 1 (1966). Often, it will be necessary for a court to look to interrelated teachings of multiple patents; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by a person having ordinary skill in the art, all in order to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue. *KSR Int'l. Co. v. Teleflex, Inc.*, No. 04-1350 (U.S. Apr. 30, 2007). Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness. *Id.* (citing *In re Kahn*, 441 F.3d 977, 988 (CA Fed. 2006)).

The obviousness rejections are predicated upon the assertions made with respect to *Carlson*. As proved above, the underlying assertions made by the examiner regarding *Carlson*'s teachings are incorrect vis-à-vis the independent claims. Specifically, *Carlson* does not teach the feature of, "obtaining application instance specific operational information identifying operational characteristics of an application instance on a computing device of the one or more computing devices, *wherein the*

application instance specific operational information includes at least one of a number of successful transactions processed by the application instance within a period of time, an application instance topology, an importance of transactions currently being processed by the application instance, an amount of time the application instance has been blocked waiting for resources, and an amount of resources consumed by the application instance," as recited in the independent claims. For this reason, *Carlson* does not teach or suggest all of the features of claim 9, at least by virtue of its dependence on the independent claim. Therefore, the rejection of claim 9 under 35 U.S.C. § 103 has been overcome.

IV. <u>35 U.S.C. § 103, Obviousness: Claims 2, 3, 12 and 13</u>

IV.A. <u>The Proposed Combination Fails to Teach All of the Features of the Dependent Claims at Least By Virtue of their Dependence on the Independent Claims</u>

The examiner has rejected claims 2, 3, 12 and 13 under 35 U.S.C. § 103 as being unpatentable over *Carlson* in view of *Jindal* et al., <u>Load Balancing in a Network Environment</u>, U.S. Patent No. 6,327,622 (December 4, 2001) (hereinafter "*Jindal*"). This rejection is respectfully traversed.

> 15. Referring to claim 2, Carlson discloses the invention substantively as described in claim 1. Carlson does not explicitly state receiving the operational information from an agent program resident on the computing device, however does discuss the use of a load balancing service including a load monitor and a load distributor (Figure 4). In analogous art, Jindal discloses another load balancing service amongst a plurality of application instances (e.g. abstract) which discloses using an agent program (i.e. individual server objects) which is capable of returning operational status information to a load balancer using a replicated monitor object 220 (col. 8, lines 23-30, 55-67). It would have been obvious to one of ordinary skill in the art to combine the teaching of Carlson in view of Jindal in order to provide an efficient method for distribution of load information of Carlson (Figures 8-9), in order for the load balancers to make an efficient determination of the weighing of the application servers.

Office Action dated March 30, 2007, p. 6

Jindal teaches a method of load balancing requests for an application among an plurality of instances of the application operating on a plurality of servers. Specifically, *Jindal* states:

> A load balancing policy is selected for distributing the client requests among the multiple servers and instances of the application and, at periodic intervals, a "preferred" server is identified in accordance with the policy. Illustratively, the selected policy reflects or specifies one or more application-specific factors or characteristics to be considered in choosing the preferred server. Client requests are routed to the preferred server until such time as a different server is preferred. A selected load balancing policy may be replaced while the application continues operating.

Jindal, col. 2, ll. 47-57

The obviousness rejections are predicated upon the assertions made with respect to *Carlson*. As proved above, the underlying assertions made by the examiner regarding *Carlson*'s teachings are incorrect vis-à-vis the independent claims. Specifically, *Carlson* does not teach the feature of, "obtaining application instance specific operational information identifying operational characteristics of an application instance on a computing device of the one or more computing devices, *wherein the application instance specific operational information includes at least one of a number of successful transactions processed by the application instance within a period of time, an application instance topology, an importance of transactions currently being processed by the application instance, an amount of time the application instance has been blocked waiting for resources, and an amount of resources consumed by the application instance*," as recited in the independent claims.

Jindal does not teach or suggest the shown deficiencies of *Carlson*. For this reason, *Carlson* and *Jindal* do not teach or suggest all of the features of claim 2, 3, 12 and 13, at least by virtue of its dependence on the independent claim. Therefore, the rejection of claims 2, 3, 12 and 13 under 35 U.S.C. § 103 has been overcome.

IV.A. <u>No Teaching, Suggestion, or Motivation Exists to Combine the References Because Each Reference Represents a Complete Solution to the Problem That Each Solves</u>

The examiner has failed to state a *prima facie* obviousness rejection against claim 2, 3, 12 and 13 because no proper teaching, suggestion, or motivation exists to combine the references. No proper teaching or motivation exists to combine the references because both *Carlson* and *Jindal* represent complete solutions to the problems each solves.

Carlson is directed to the problem of improving access times and process redundancy for application servers. For example, *Carlson* provides that:

> Application servers offer significant advantages over previous approaches to implementing web applications, such as using common gateway interface (CGI) scripts or programs. FIG. 1 illustrates a typical architecture for a web application utilizing CGI scripts or programs. The client computer running a web browser 10 may reference a CGI program 18 on the web server 14, e.g., by referencing a URL such as "http://server.domain.com/cgi-bin/myprogram.pl" over an Intranet or the Internet 12. Generally, the CGI program runs on the web server itself, possibly accessing a database 20, e.g. in order to dynamically generate HTML content, and the web server returns the output of the program to the web browser. One drawback to this approach is that the web server may start a new process each time a CGI program or script is invoked, which can result in a high processing overhead, impose a limit on the number of CGI programs that can run at a given time, and slow down the performance of the web server. In contrast, application servers typically provide a means for enabling programs or program components that are referenced via a URL to run on a separate computer from the web server and to persist between client invocations.
>
> Another common drawback of previous web application design models, such as the use of CGI programs, is related to data access. For example, if a CGI program needs to

> access a database, the program typically opens a database connection and then closes the connection once it is done. Since opening and closing database connections are expensive operations, these operations may further decrease the performance of the web server each time a CGI program runs. In contrast, application servers typically provide a means to pool database connections, thus eliminating or reducing the need to constantly open/close database connections. Also, data access in CGI programs is generally coded at a relatively low level, e.g., using a specific dialect of SQL to access a specific type of database. Thus, portions of the application may need to be recoded if the database is replaced with a new type of database. Application servers, on the other hand, may provide a database service for applications to utilize as an interface between the application and the database, which can serve to abstract the application from a particular type of database.

Carlson, col.1, l. 31-col. 2, l. 5

Carlson solves this problem by caching JavaServer Page.TM. (JSP) component responses. The JSP components may also be configured to utilize a cluster of application servers in which requests from the client computer(s) are distributed across different application servers. Specifically, *Carlson* provides that:

> The execution of JSP components may be managed by a JSP engine process running on the application server. When a request referencing a JSP is received from a client computer, the JSP engine may first check a JSP response cache to determine whether a valid JSP response satisfying the request is present. A JSP request may comprise various types of attributes, such as variable name and value pairs, that are matched against criteria sets associated with each response entry. If a matching cached response is found, then the response may be retrieved and immediately streamed back to the client. Otherwise, the referenced JSP may be executed.
>
> Each JSP file may comprise code for specifying caching criteria for the JSP. This code may cause the output (or response) for the JSP to be cached once the JSP execution is complete, where the cache entry is associated with the specified caching criteria. The caching criteria may then be used to match against attributes of future JSP requests, as described above. The caching criteria may include a "time" argument indicating the number of seconds for which the cached response should be considered valid, a criteria string specifying attribute criteria to use in determining whether or not the cached response may be used to satisfy a request, etc. In particular embodiments, any of various types of cache criteria may be supported, as described below. The caching criteria may be specified in the JSP code in various ways, e.g., by a method invocation, by using markup tags, etc.

Carlson, col.5, ll. 40-67

On the other hand, *Jindal* is concerned with the problem of improving load balancing among various instances of an application. For example, *Jindal* provides that:

> present load balancing techniques are also limited in scope. For example, the techniques described above are designed for replicated services only and, in addition, only consider the operational status or characteristics of the servers hosting the replicated service, not the service itself. In other words, present techniques do not allow load balancing among instances of an application program or, more generally, the collection or consideration of information concerning the status of individual instances of applications or services executing on multiple servers.

Jindal, col.2, ll. 28-38

Jindal solves this problem by providing different load balancing policies based on individual monitoring objects for collecting information about each instance. The load balancing policies are then updated based on this information. Specifically, *Jindal* provides that:

> A load balancing policy is selected for distributing the client requests among the multiple servers and instances of the application and, at periodic intervals, a "preferred" server is identified in accordance with the policy. Illustratively, the selected policy reflects or specifies one or more application-specific factors or characteristics to be considered in choosing the preferred server. Client requests are routed to the preferred server until such time as a different server is preferred. A selected load balancing policy may be replaced while the application continues operating.

Jindal, col.2, ll. 47-56

> Depending upon the selected policy, status objects (e.g., agents, modules or other series of executable instructions) are configured to collect these various pieces of information from each instance of the application that is being load-balanced (and/or its server). Status objects in one embodiment of the invention thus retrieve application-specific information (e.g., number and/or type of pending client requests) and/or information concerning a server's general status (e.g., its distance from another network entity). Illustratively, each instance of a load-balanced application is associated with its own status object(s). In one embodiment of the invention multiple status objects having different functions are associated with one instance.
>
> Each instance of the application (or, alternatively, each server hosting an instance of the application) is also associated with an individual monitor object or IMO (e.g., another object, module or series of executable instructions). Each IMO invokes and stores information from one or more status object(s) collecting information concerning an instance of the application. In one embodiment of the invention each IMO is configured to interact with a single status object; in an alternative embodiment multiple status objects are associated with an IMO. In addition, in one embodiment of the invention an IMO interfaces directly with its status object(s); in another embodiment each status object stores its application-specific information for retrieval by the IMO.

Jindal, col.3, ll. 1-26

Jindal has no need to modify the disclosed method. *Jindal* provides a complete solution to the problem that *Jindal* addresses.

Because each reference provides a complete solution to the problem that each reference represents, one of ordinary skill would have no reason to combine or otherwise modify the references. For this reason, no teaching, suggestion, or motivation exists to combine the references to achieve the invention of claims 2, 3, 12 and 13. Accordingly, the examiner has failed to state a *prima facie* obviousness rejection against claims 2, 3, 12 and 13.

V. Conclusion

It is respectfully urged that the subject application is patentable over *Carlson* and *Jindal* and is now in condition for allowance. The examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: July 2, 2007

Respectfully submitted,

/Brandon G. Williams/

Brandon G. Williams
Reg. No. 48,844
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants